# Processor & I/O Virtualization

**Don Banks, Phil Winterfield**

**April 22nd, 2008**

# A Look at Current Virtualization Technology

- ***Virtualization Use Models***
- ***Virtualization Technology  - Definitions & Architecture***
  - Hardware-assisted Processor & I/O virtualization supporting VM environments
- ***Processor Vendors - where are they at w.r.t. virtualization?***
  - POWER (IBM, PA Semi, Freescale)
  - IA32/x86 (Intel, AMD)
  - UltraSPARC (Sun Niagara2)
  - MIPS-64 (Cavium)
- ***Early Prototype Experience***
  - Sun Niagara2, Intel IA32
  - Xen
  - IOS, Linux

# Virtualization Use Models

# 3 Base Use Models

- **Workload *Consolidation*: increased resource utilization**
  - Application & hardware consolidation
  - Multiple different simultaneous execution environments (SMP, UP, operating systems)
  - Collocation - loosely coupled communicating VMs distributed on-chip
  - 3rd party application hosting (not porting)
  - Load balancing
- **Workload *Isolation*: security and fault domain isolation**
  - Containerization (i.e., VMs)
    - Specialized kernels (e.g., secure, boot loader, TPM )
    - Sandboxing
    - Protected application spaces
    - Increased RAS
  - Hypervisor manages container resource utilization, access & QoS
  - Hardware provides chain of trust & isolation mechanisms
- **Workload *Migration*: Dynamic - uses the network; requires network intelligence/state**
  - Migrate work to where it needs to be either automatically according to policy or on demand
    - Green initiative - within the box and within the data center: power & heat management
    - Workload balancing
    - Hardware maintenance & upgrades (HA)

# Some Product Development Virtualization Use Models

- **Packaging/delivery (“containers”)**
  - Smaller, self-contained, specialized VMs
  - Less integration (integrate via protocol) - limits dependency domain between features/products
  - Less hardware dependencies
  - Simpler maintenance (fewer causes, fewer dependencies)
  - Hosting vs. porting for 3rd party apps
  - = faster TTM
- **Testing**
  - Fewer components in package to test, easier to validate (fewer dependencies)
  - Faster test cycles
  - = faster TTM
- **Development**
  - Individual VMs (anytime, anywhere); many users/many VMs per user/many VMs per hardware platform
  - No/limited need to schedule hardware
  - Better debugging for unit test/development; network testing in a box
  - No fault “spill” to other VMs – isolated fault domains; sharing easy
  - = decreased development/debug time
- **Reliability, Availability, Serviceability, Security**
  - VM isolation and simplified container VMs provide better RAS - limits fault domain
  - Specialized & isolated VMs + secure hardware features provides more secure implementations
  - Chain of trust and secure VMs allow for trusted [licensed] software distribution on demand
  - = less bugs, secure operations
- **Increased parallelism made available**
  - Multiple VMs managed by policy-based hypervisor; resource utilization enforced by hypervisor
  - Natural parallelism: no need for special programming (for decoupled and loosely coupled applications)
- **Allows use of COTS hardware (where appropriate)**
  - e.g., ATCA/uTCA chassis & power, cooling

# Virtualization Technical Overview

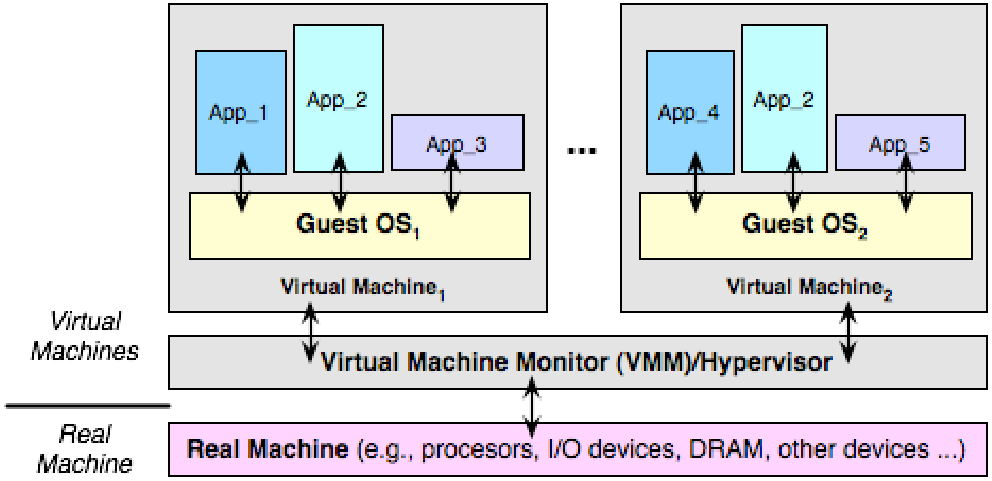
# Microarchitecture Evolution

- Single-core processors no longer scale with clock frequency
  - Power & heat increase with process shrink & clock speedup
    - Expect only 5-10% clock speedup in future generations
    - Lots of slower cores; no new single-core devices (except at very low end - e.g. Intel “Atom”)
- But, Moore’s Law still applies
  - How to use all those transistors?
    - Multi-core processors are the direction of all major vendor architectures to address these problems (e.g., POWER (IBM, PA Semi, Freescale), IA32/x86 (Intel, AMD), UltraSPARC (Sun Niagara2), MIPS-64 (Cavium))
    - 2, 4, 6, 8, …, 16, … 80 core (also hardware multithreading - CMT, SMT)
    - Hardware-supported virtualization incorporated into most designs
- How to utilize all those cores?
  - Develop new highly-threaded software
    - New software, new design paradigm, new tools, …
    - But, not all problems lend themselves to highly threaded solutions
    - What to do with the [large] existing body of single-threaded code?

  Or

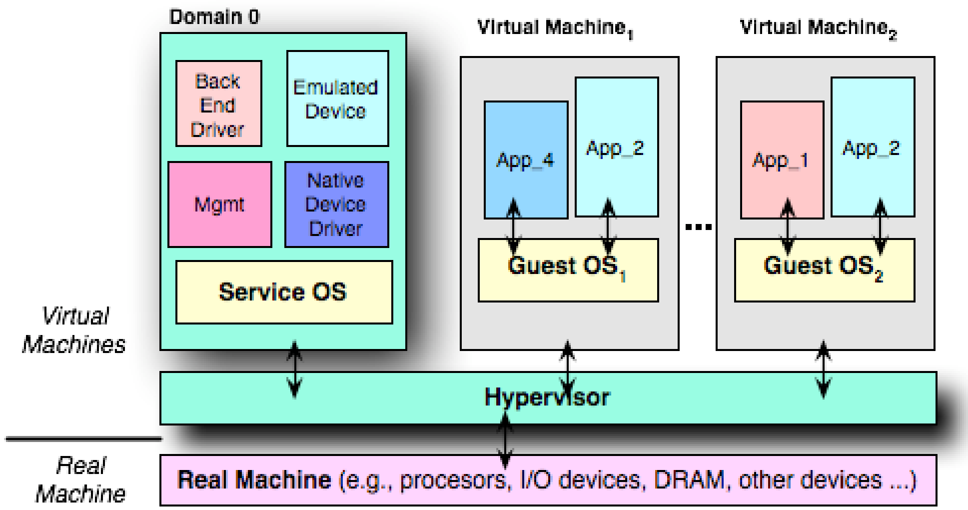
  - Run many existing single-threaded and available multi-threaded applications in the same machine → virtualization

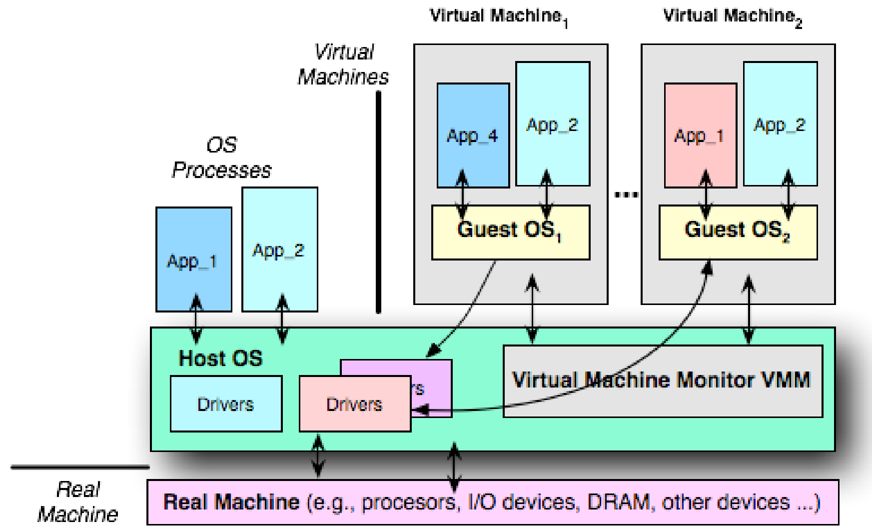
# General Virtual Machine Model




- ***Hypervisor/Virtual Machine Monitor (VMM) and Virtual Machines (VMs)***
  - Sits directly on the "bare" hardware; virtualizes/arbitrates hardware resources
    - – Isolates hypervisor and VMs from each other using hardware and hypervisor software mechanisms (fault domain isolation)
      - VM ("partition" or "domain") contains an OS, drivers and applications
      - OS cannot directly access real hardware that affects the state of the machine
      - OS may be "fully virtualized" (no changes required), or "paravirtualized" (hypervisor aware and modified to interface directly to hypervisor via "hcalls")
  - Manages provisioning, creation, and scheduling of VMs and guarantees QoS
    - – Multiplexes access to shared resources using provisioned policy definitions
    - – For performance, some cores and/or devices may be dedicated

# Hypervisors/VMMs

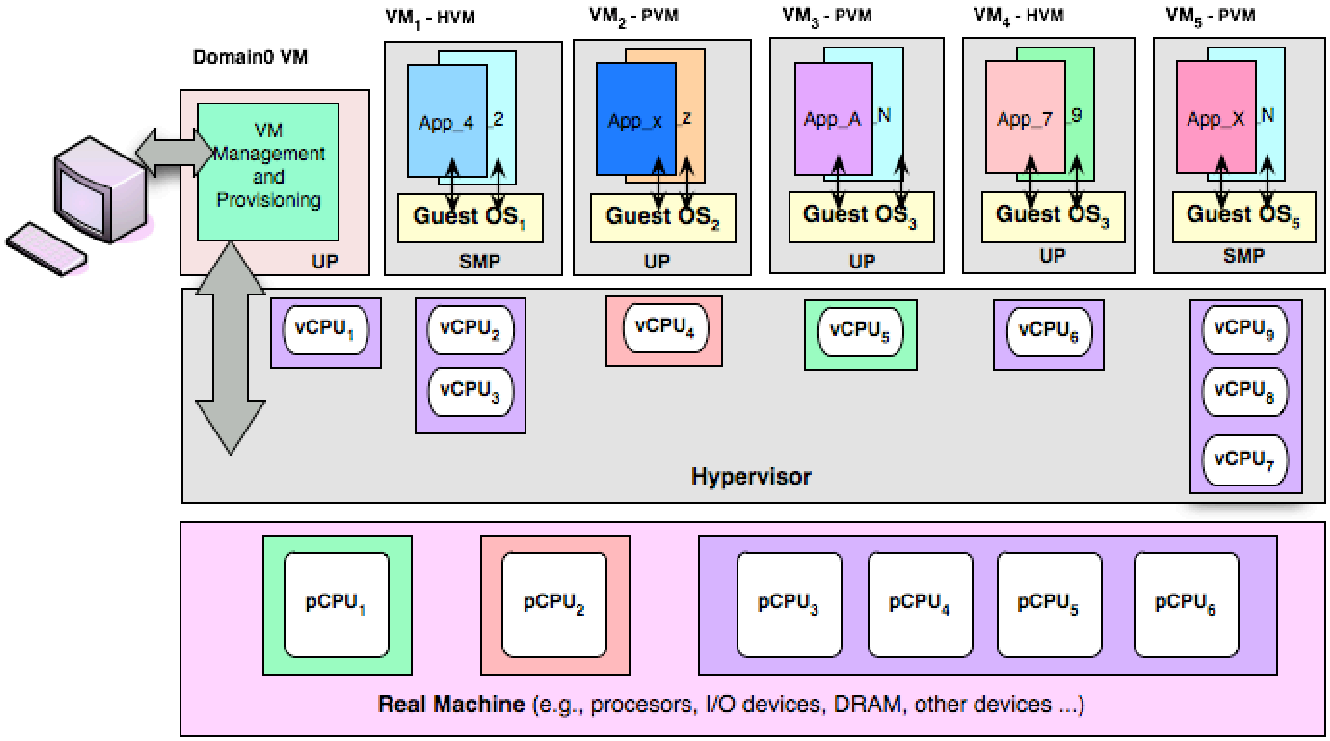



- ***Native ("thin") hypervisor (e.g., Xen)***
  - Utilizes a "privileged" VM ("domain 0") to augment hypervisor → hypervisor less complex
    - Management, provisioning, and monitoring
    - Emulation and Virtualized drivers → possible greater I/O overhead, higher latency
    - dom0 can fail (or be upgraded), restart without affecting hypervisor or VMs (service interrupted to served devices)
  - Runtime footprint: hypervisor ~100MB + dom0 ~100-250MB = 200-350MB
    - ~8GB machine, 4x2GB 32-bit PV guests
  - Hypervisor simpler → more reliable, less maintenance → more available

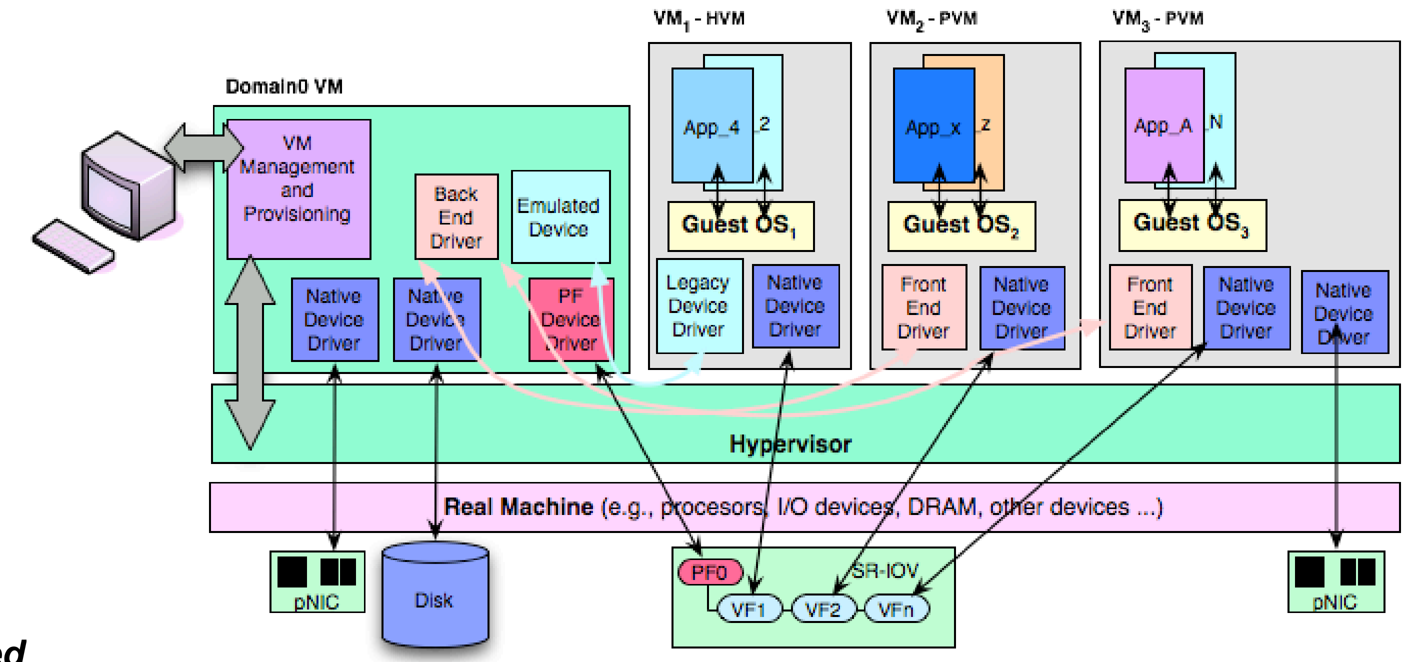
- ***OS-hosted ("thick") hypervisor (VMM) (e.g., VMware, kvm)***
  - Full OS process environment + VMM
    - VMM is component of OS to support VMs; shares resources with process environment
    - All native emulation/virtual drivers in OS → less I/O overhead, lower latency, more complexity
    - OS schedules processes and VMM; VMM schedules VMs
    - VMM failure (or upgrade) = system failure
  - Runtime footprint: ESX Server VMM ~700MB
    - ~8GB machine, 4x2GB 32-bit VMs
  - VMM more complex → less reliable, more maintenance → less available

# Virtual CPU Scheduling




- ***Virtual CPUs (vCPU) mapped to physical CPUs (pCPU) by hypervisor***
  - Dedicated
    - vCPU mapped directly to a specific pCPU (1-to-1 correspondence)
    - No need for scheduling by hypervisor
    - Near native performance
  - Shared (depends on hardware architecture)
    - More than one vCPU mapped to a pCPU (typically a pool of pCPUs)
    - VMs need to be scheduled on and off the pCPUs by the hypervisor (fair share, earliest deadline first, credit scheduler - weight & cap)
    - Possible lower performance, higher latency (but may not matter)
- ***Supports SMP and UP models***
- ***Requires virtualization of shared and protected hardware resources***
  - Requires hardware support to allow hypervisor to do this safely and efficiently
  - e.g. clocks, interrupt controller, shared FP device, other SoC devices, registers/tables req'd by hypervisor …

# Device Models




- ***Emulated***
  - Supports Legacy guest device drivers; no changes needed
  - Communicates with emulation driver in dom0 which uses native device driver there for actual I/O
  - Lowest performance
  - Can be shared
- ***Split-driver (Front-end/Back-end) or Virtual device driver***
  - Requires guest paravirtualization
  - Front-end in guest VM, back-end in dom0; uses native device driver there for actual I/O
  - Better performance
  - Usually shared
- ***Dedicated driver***
  - Requires guest/driver paravirtualization (e.g., Xen HVM device drivers)
  - Requires hardware IO virtualization to remain safe (e.g., IOTLB, DMA & I/O interrupt remapping, multiple device queues)
  - Near native performance
  - (Also PCIe SR-IOV devices and ATS)

# Vendor Status

# UltraSPARC & IA32

- ***Sun4v Virtual Machine Architecture***
  - **Niagara2 (UltraSPARC T2)**
    - (hyper-privileged, privileged, user) processor modes
    - “always on” hypervisor (Sun proprietary firmware)
      - Requires OS to be paravirtualized
      - Secure boot
    - Thread is unit of virtualization (no shared/fixed # virtual cpus) - Chip Multi-Threading (CMT)
    - Logical Domains (LDOMS) 1.0 (“LDOM” = VM)
    - Solaris must be Control/Service domain (I.e., LDOM0); no paravirtualized Linux for guest domains
- ***Intel Virtualization Technology for IA32/x86 (VT-*)***
  - **IA32 architecture (no absolute owner)**
    - Boots in legacy mode; “hypervisor mode” enabled/disabled by (VMXON/VMXOFF)/(VMRUN) instructions
    - Many software hypervisors/VMMs available (selected by user)
      - Most support both fully virtualized and paravirtualized models
    - Intel and AMD implementations similar but not compatible
    - Core is unit of virtualization (shared & non-shared virtual cpus) - SMT (per core)
  - **Intel VT-* technology**
    - *Processor*: VT-x (Virtualization Machine eXtensions - VMX)
    - *Chipset*: VT-d (Virtualization Technology for Directed IO) DMA and Interrupt remapping, PCIe ATS (Address Translation Services)
    - *I/O Device*: VMDq (Virtual Machine Device Queues), IOAT (I/O Acceleration Technology), L2 sorter/classifier, PCIe SR-IOV (Single Root I/O Virtualization) - virtual devices presented by physical device
    - Migration support: Flexmigration
    - Security and TPM support: TXT, sealed storage, protected execution/memory space,
    - Using Xen (unstable tree to support Intel VT-* functionality) + paravirtualized Linux in dom0
  - **AMD Virtualization (AMD-V) Secure Virtual Machine (SVM) technology (a.k.a. Pacifica)**
    - *Processor*: AMD-V (AMD Virtualization)
    - *Chipset*: DMA remapping; interrupt remapping ; PCIe ATS (Address Translation Services)
    - *I/O Device*: PCIe SR-IOV (Single Root I/O Virtualization) - virtual devices presented by physical device
    - Security and TPM support: SVM SKINIT for trusted VMM boot using a Secure Loader (64KB) (non-trusted → trusted mode)

# POWER & MIPS/Cavium

- ***Power***
  - **POWER6 architecture** owned by power.org, Licensed by members
    - (hypervisor, supervisor, user) processor modes
    - Core is unit of virtualization (shared & non-shared virtual cpus) - SMT (per core)
    - Logical Partitioning (LPARs), Virtual I/O
    - Migration support: Partition Mobility
  - **IBM**
    - “always on” hypervisor (firmware) - proprietary Advanced Processor Virtualization (APV) Hypervisor
    - Requires OS to be paravirtualized
    - Proprietary OS in IOVM & partition 0; paravirtualized Linux in guest domains + IBM proprietary OSs
    - Secure boot
  - **PA Semi**
    - Power Book III E Server; inherits hardware virtualization support through POWER ISA def’n (power.org)
    - Hard partitioning
    - No solid plans for a hypervisor
    - Availability ??
  - **Freescale**
    - Power Book III E Embedded; inherits hardware virtualization support through POWER ISA def’n (power.org)
    - Hard partitioning
    - Developing own hypervisor targeted at “embedded” (in progress)
    - Hardware based security to create “root of trust” for boot
    - 8578 CoreNet Product availability: 2Q/3Q09 (alpha/beta parts projected)
- ***MIPS/Cavium***
  - Plans unclear
    - No processor virtualization; some form of I/O virtualization; 3rd party hypervisor support (no MIPS architecture def’n)

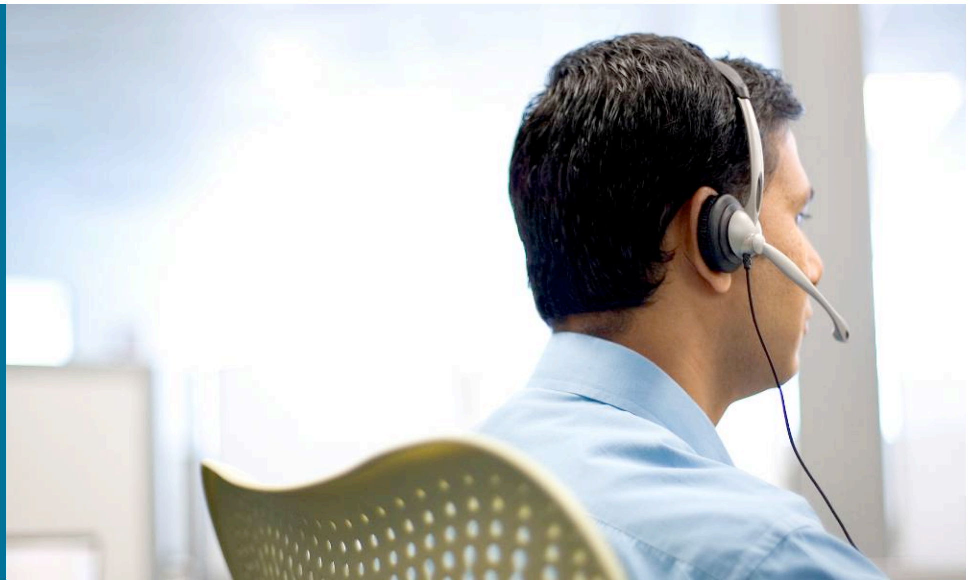
# Hardware Virtualization Features To Look For

- ***Processor Virtualization***
  - Hypervisor processor mode
    - Hardware protected execution environment
  - Virtual Machine management instruction set
    - VM context switching instructions to manage protected VM state in hardware
    - Hypervisor call instruction
  - Instructions/hardware to minimize VM exit conditions
    - Fine grained control over guest VM exit causes (usually bit maps)
    - Protected register virtualization
    - TLB tagging (with ASID/VMID/VPID)
    - Hypervisor preemption timer (guest scheduling control)
    - Guest idle detection (guest scheduling control)
    - 2-level Page Tables (eliminates shadow page table exit overhead)
  - Efficient VM-to-VM messaging support
- ***I/O Virtualization***
  - IOMMU/IOTLB
    - DMA remapping (memory protection & isolation)
    - Interrupt remapping
- ***Device Virtualization - processor offload (on chip devices too)***
  - PCIe SR-IOV & ATS support
  - Multiple, weighted round robin-based, queues in packet processing devices (sorting & grouping packets in & out)
  - Low latency interrupts
  - Direct cache access
- ***Hypervisor***
  - Supports hardware virtualization functionality of target architecture
  - Management/provisioning interface
  - Direct device assignment
  - Shared/dedicated resource capability
  - Appropriate scheduling controls (for shared resources)
- ***Security (works with virtualization)***
  - Hardware rooted trust - secure boot & support for TPM in secure VM

# Summary

- **Many possible ways to exploit both multi-core and virtualization technology to our advantage, but …**
  - Virtualization is a fundamental technology, not a product
  - Not all solutions/products can or should use virtualization
  - Microarchitecture & hypervisor architecture *are* important
    - "virtualization solution A" ≠ "virtualization solution B" ≠ "virtualization solution C" …
  - Virtualization is *not* free;
    - It costs resources (memory, processor)
    - May cost latency (configuration dependent - both processor & I/O)
  - Additional management complexity due to VM provisioning and management
    - Management systems vary in capability and ease of use across vendor solutions
    - Except for lowest end products, have to teach customers how to provision VMs (or automate)
  - This is *not* new - there is 40+ years of history and experience to learn from …

# Prototype Description & Status



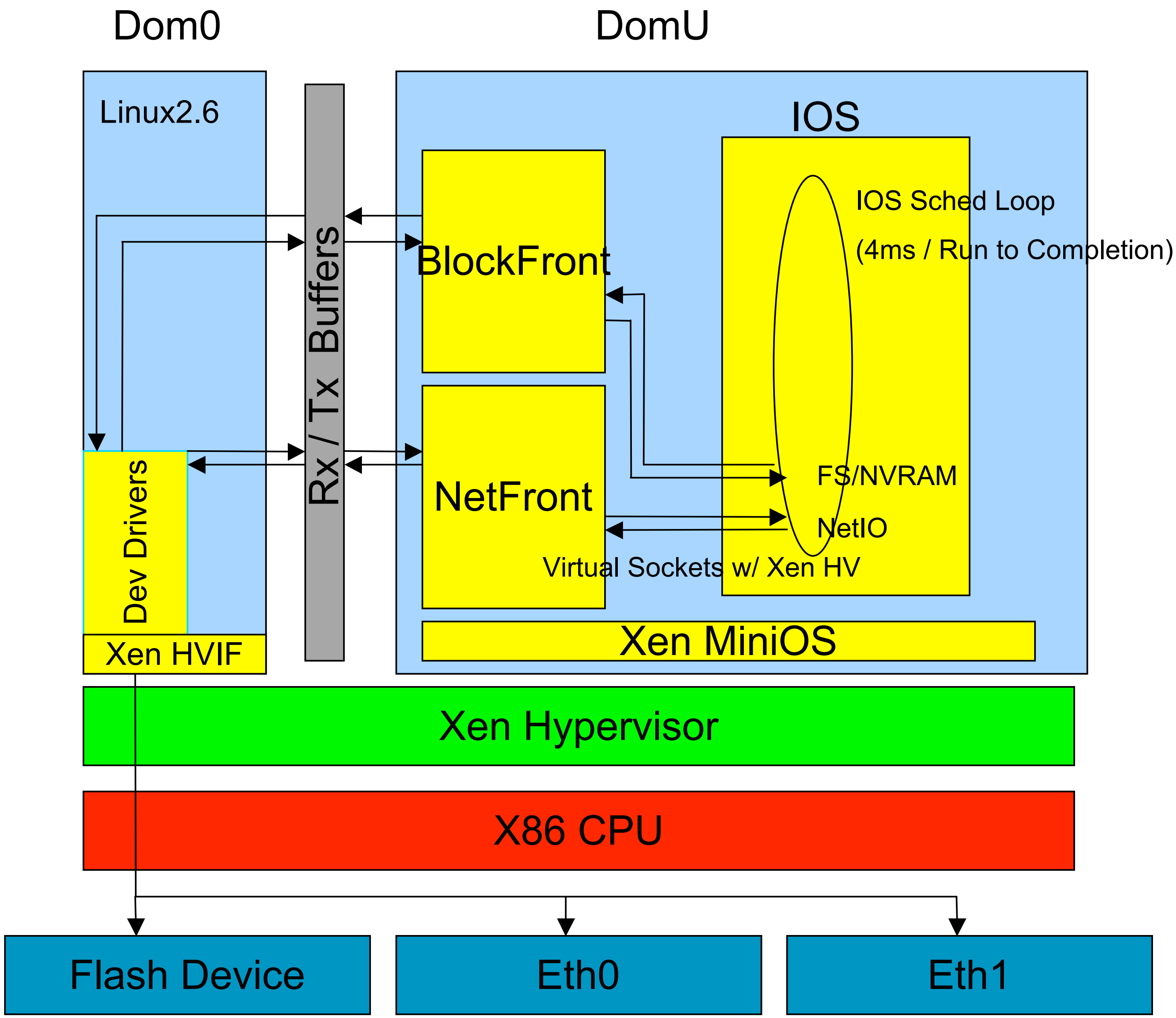
# Sun4v Niagara2

- Project to build a paravirtualized version of IOS and use a paravirtualized Linux to run router/switch workloads on top of the Sun4v hypervisor and the Niagara-2 chipset
  - Delays due to defective h/w (4 weeks)
  - Sun4v is 64-bit and required 64-bit IOS
  - The 64-bit IOS project was (and remains) unfinished
  - Advanced UltraSPARC features required unreleased gcc3.4 version for IOS builds
  - Unlike Xen, Sun offers no SDK, detailed documentation or proto startup code base
    - Hypervisor docs & APIs unclear, some features undocumented
  - NDPS (Sun code) based prototyping effort stopped because of proprietary code issues.
  - Developers hard to get to; no internal Linux support
  - Solaris required for Control & Service domains (No Linux support)
  - (More detailed project summary available)
- **Conclusions:**
  - Sun4v/Niagara2 is not:
    - Ready for “prime time”
    - $ Competitive w/ Intel or Power virtualization solutions
    - Architecturally suited to low latency single threaded workloads

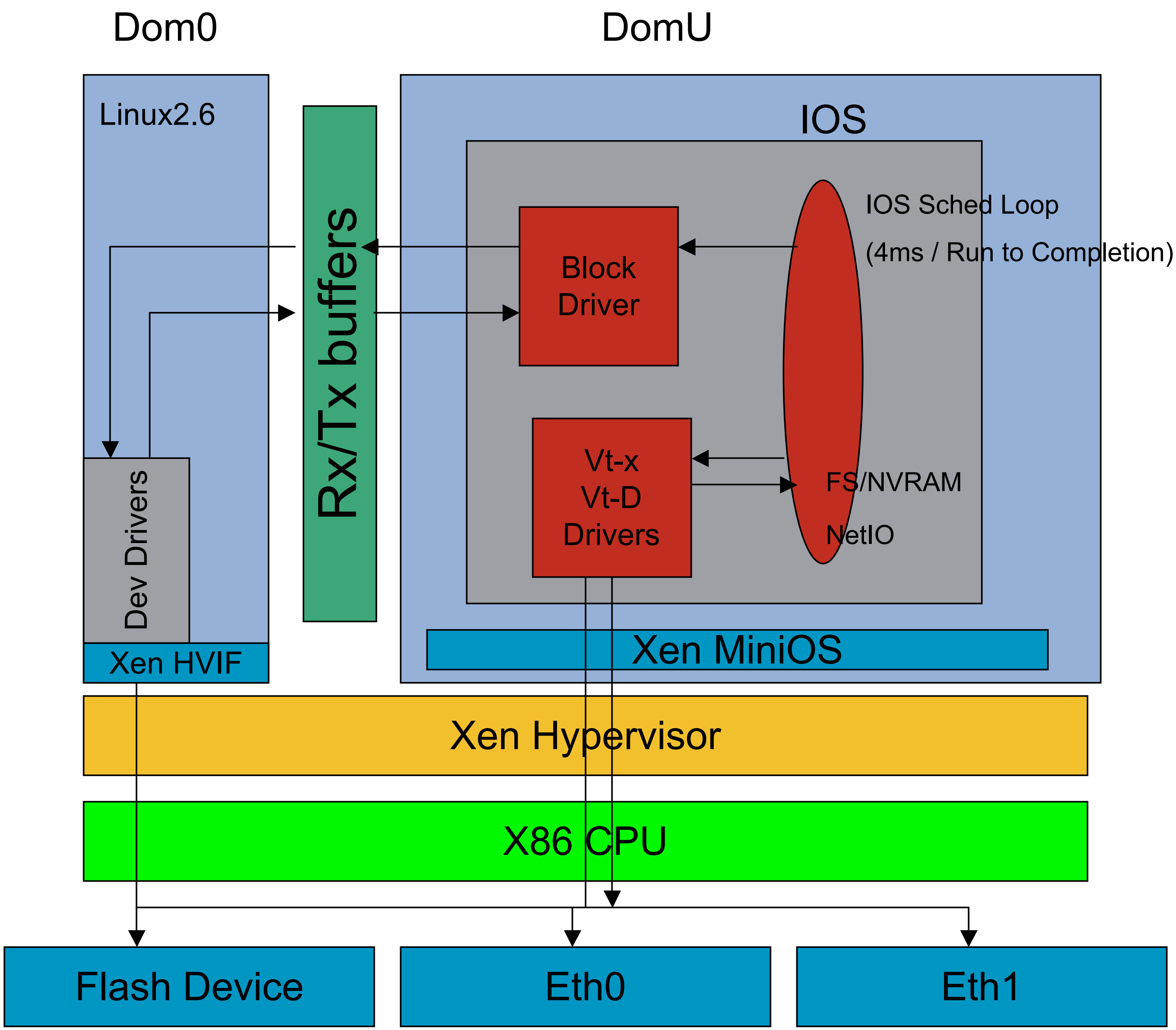
# Intel VT-* with Xen

- Building a Paravirtualized IOS to run on Intel VT hardware and Xen
    - Collaborative effort w/ Intel and their Xen internal development group
    - Many instabilities w/ Vt-x and Vt-d hardware (3 months and counting)
    - Reverted to Vt-x h/w for stability reasons (hardware, firmware, Xen)
    - Intel expects Vt-d (Shofner - server) boards available in 1-2 weeks
    - Began bringup and unit test 4 weeks ago
    - ARTG to add developers to effort
    - Near term development effort to focus on IOS Vt-d driver, NIC with VMDq & L2 classifier/sorter (dedicated I/O device assigned to IOS and/or Linux)
- Conclusions:
    - Not as far along as white papers and slides would indicate
    - Good customer support and [internal developers] actively developing and working on Xen
    - Hardware & hypervisor software will support full processor & I/O virtualization
        - Performance TBD

# Paravirtualized IOS on Xen Prototype (Phase 1: Shared Devices)



# Paravirtualized IOS on Xen Prototype (Phase 2: Dedicated Devices)



# Tools & Test

- Tools
  - 64-bit GCC4.2.1 compiler required for sun4v
    - In progress within Cisco but not yet released
    - Modified build environment (by us)
  - Intel Bi-endian compiler + build environment required for VT-x
    - From previous x86 projects in Cisco
  - Xen 3.3 (for Intel) with VT-* Support
    - From Intel
  - PowerPC requirements TBD (by vendor)
- Test Infrastructure
  - Smartbits w/ 2x1GB and 2x10GB ports & network traffic scripts (from others)
  - Representative workloads from ARTG and others

# Wiki Reference for Virtualization

- http://tags/twiki/bin/view/VirtCenter/WebHome
  - Virtualization whitepaper, "*Current Processor, I/O and Hypervisor/VMM Virtualization Technology: a Survey*"
  - Other virtualization reference papers, documents, and public vendor docs

- Looking for your advice & feedback